图文 023、一步一图：那JVM老年代垃圾回收器CMS工作时，内部又干

详情 评论

一步一图

JVM老年代垃圾回收器工作时

内部又干了些啥?

给大家推荐一套质量极高的Java面试训练营课程：

作者是中华石杉，石杉老哥是我之前所在团队的 Leader，骨灰级的技术神牛！

大家可以点击下方链接，了解更多详情，并进行试听：

21天互联网Java进阶面试训练营（分布式篇）

重要说明：

最近不少同学留言反馈，说希望建立一个微信群，供大家进行JVM专栏的学习交流。

这个提议非常好，不过管理微信群是一件挺费时的事儿，我平时工作较忙，实在抽不出时间来进行群管理.

正好石杉老哥的面试训练营建了微信交流群，并且还请了不少一线大厂的助教。

因此跟石杉老哥商量了一下，决定厚着脸皮“鸠占鹊巢”。购买了我JVM专栏的小伙伴，可以加入石杉老哥的微信群，在群里讨论交流技术。

如何加群，请参见文末（注：如果之前已经加过的，就不要重复加群了）

1、前文回顾

本文我们就要进入最核心的老年代垃圾回收环节了，之前的文章大家看过之后对JVM的核心原理都有一定的了解了，年轻代的垃圾回收机制也都很清楚了，其实年轻代的垃圾回收通过复制算法来，还是比较简单的。

大家心里最希望的，就是对象都分配在新生代的Eden区，然后每次垃圾回收之后，存活对象都进入Survivor区，然后下一次垃圾回收后的存活对象都进入另外一个Survivor区。

这样几乎很少很少的对象会进入老年代里去，也就几乎不太会触发老年代的垃圾回收了。

但是理想很丰满，现实很骨干。其实大家想想，你们在写代码的时候，有谁会考虑垃圾回收啥的？不会有人考虑这个吧，就是不停的狂写代码，然后直接上线部署，根本没多少人会考虑自己的代码对垃圾回收的影响。

最多有经验的工程师上线之前，通过我们之前的案例讲解的方法估算一下系统的内存压力以及垃圾回收的运行模型，然后合理设置一下内存各个区域的大小，尽量避免太多对象进行老年代里去。

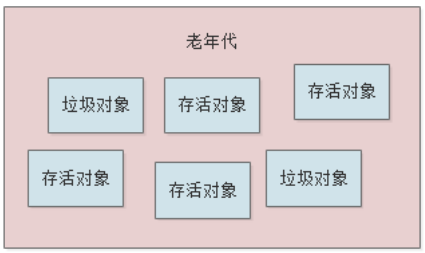
但是真实情况是，线上系统很可能就会因为各种各样的情况，导致很多对象进入老年代，然后甚至频繁触发老年代的Full GC。

之前我们用案例给大家演示过很多这种情况，比如说Survivor区太小，容纳不了每次Minor GC后的存活对象，导致对象频繁进入老年代，频繁触发老年代Full GC。

类似的情况其实很多，所以，咱们不能过于理想化的期待永远没有老年代GC，还是要对老年代的垃圾回收器是如何回收的，有一个了解和认识。

2、CMS垃圾回收的基本原理

一般老年代我们选择的垃圾回收器是CMS，他采用的是标记清理算法，其实非常简单，就是先用之前文章里讲过的标记方法去标记出哪些对象是垃圾对象，然后就把这些垃圾对象清理掉，如下图所示。

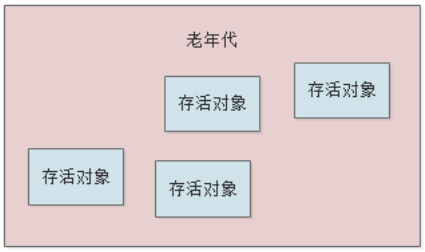


上面图里是一个老年代内存区域的对象分布情况，现在假设因为老年代内存空间小于了历次Minor GC后升入老年代对象的平均大小，判断Minor GC有风险，可能就会提前触发Full GC回收老年代的垃圾对象。

或者是一次Minor GC后的对象太多了，都要升入老年代，发现空间不足，出发了一次老年代的Full GC。

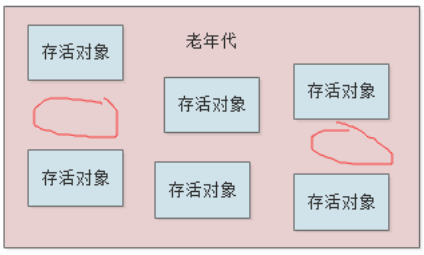
总之就是要进行Full GC了，此时所谓的标记-清理算法，其实就是我们之前给大家讲过的一个算法，先通过追踪GC Roots的方法，看看各个对象是否被GC Roots给引用了，如果是的话，那就是存活对象，否则就是垃圾对象。

先将垃圾对象都标记出来，然后一次性把垃圾对象都回收掉，如下图。



这种方法其实最大的问题，就是会造成很多内存碎片

大家看下图的红圈处就是所谓的内存碎片，这种碎片不大不小的，可能放不小 任何一个对象，那么这个内存就被浪费了，之前我们聊过这个问题。



这就是CMS采取的“标记-清理”算法。

3、如果Stop the World然后垃圾回收会如何?

现在大家思考一个问题，假设要先“Stop the World”，然后再采用“标记-清理”算法去回收垃圾，那么会有什么问题?

之前文章也说过了，如果停止一切工作线程，然后慢慢的去执行“标记-清理”算法，会导致系统卡死时间过长，很多响应无法处理。

所以CMS垃圾回收器采取的是垃圾回收线程和系统工作线程尽量同时执行的模式来处理的。

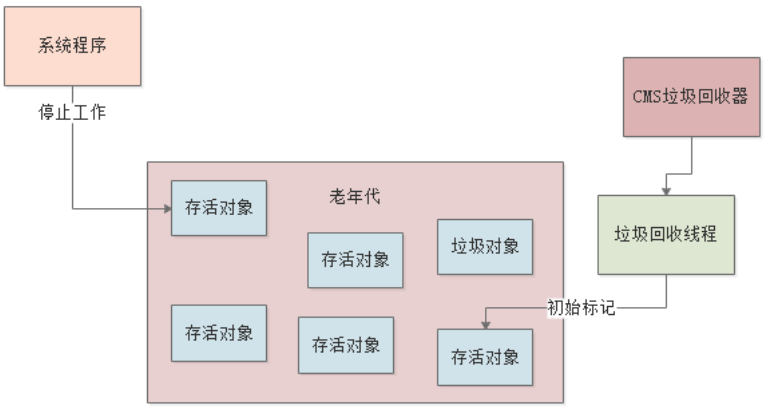
4、CMS如何实现系统一边工作的同时进行垃圾回收?

CMS在执行一次垃圾回收的过程一共分为4个阶段:

初始标记
并发标记
重新标记
并发清理

我们一点一点来看。

首先，CMS要进行垃圾回收时，会先执行初始标记阶段，这个阶段会让系统的工作线程全部停止，进入“Stop the World”状态，如下图。



所谓的“初始标记”，他是说标记出来所有GC Roots直接引用的对象，这是啥意思呢?

比如下面的代码。

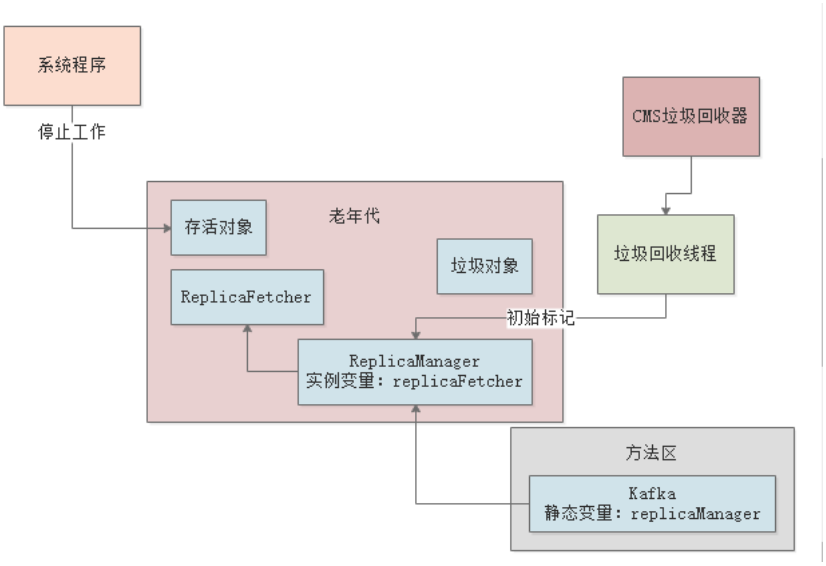
```
public class Kafka {
    private static ReplicaManager replicaManager = new ReplicaManager();
}
public class ReplicaManager {
    private ReplicaFetcher replicaFetcher = new ReplicaFetcher();
}
```

在初始标记阶段，仅仅会通过“replicaManager”这个类的静态变量代表的GC Roots，去标记出来他直接引用的ReplicaManager对象，这就是初始标记的过程。

他不会去管ReplicaFetcher这种对象，因为ReplicaFetcher对象是被ReplicaManager类的“replicaFetcher”实例变量引用的

之前说过，方法的局部变量和类的静态变量是GC Roots。但是类的实例变量不是GC Roots。

如下图所示。



所以第一个阶段，初始标记，虽然说要造成“Stop the World”暂停一切工作线程，但是其实影响不大，因为他的速度很快，仅仅标记GC Roots直接引用的那些对象罢了。

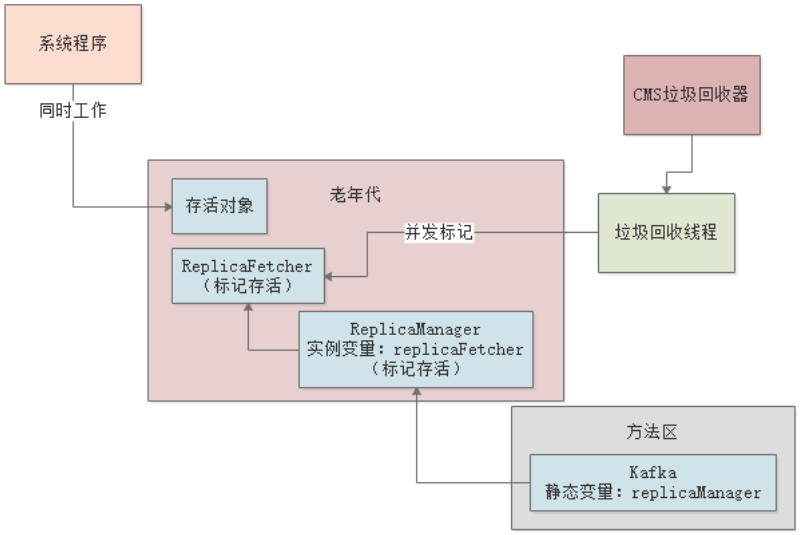
接着第二个阶段，是并发标记，这个阶段会让系统线程可以随意创建各种新对象，继续运行

在运行期间可能会创建新的存活对象，也可能会让部分存活对象失去引用，变成垃圾对象。在这个过程中，垃圾回收线程，会尽可能的对已有的对象进行GC Roots追踪。

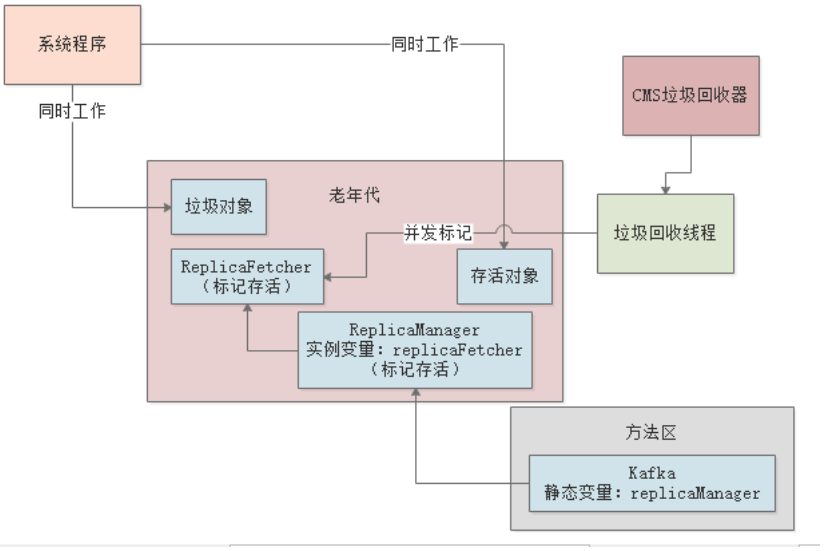
所谓进行GC Roots追踪，意思就是对类似“ReplicaFetcher”之类的全部老年代里的对象，他会去看他被谁引用了？

比如这里是被“ReplicaManager”对象的实例变量引用了，接着会看，“ReplicaManager”对象被谁引用了？会发现被“Kafka”类的静态变量引用了。

那么此时可以认定“ReplicaFetcher”对象是被GC Roots间接引用的，所以此时就不需要回收他。如下图所示。



但是这个过程中，在进行并发标记的时候，系统程序会不停的工作，他可能会各种创建出来新的对象，部分对象可能成为垃圾，如下图所示。



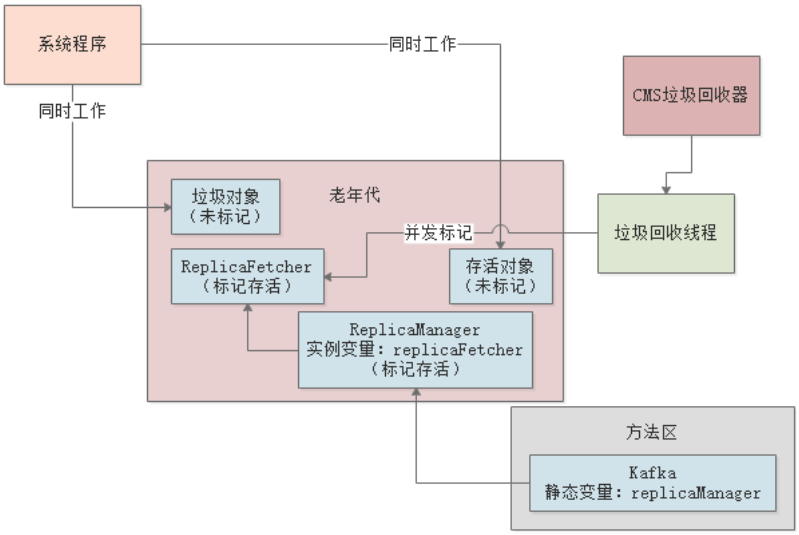
第二个阶段，就是对老年代所有对象进行GC Roots追踪，其实是最耗时的

他需要追踪所有对象是否从根源上被GC Roots引用了，但是这个最耗时的阶段，是跟系统程序并发运行的，所以其实这个阶段不会对系统运行造成影响的。

接着会进入第三个阶段，重新标记阶段

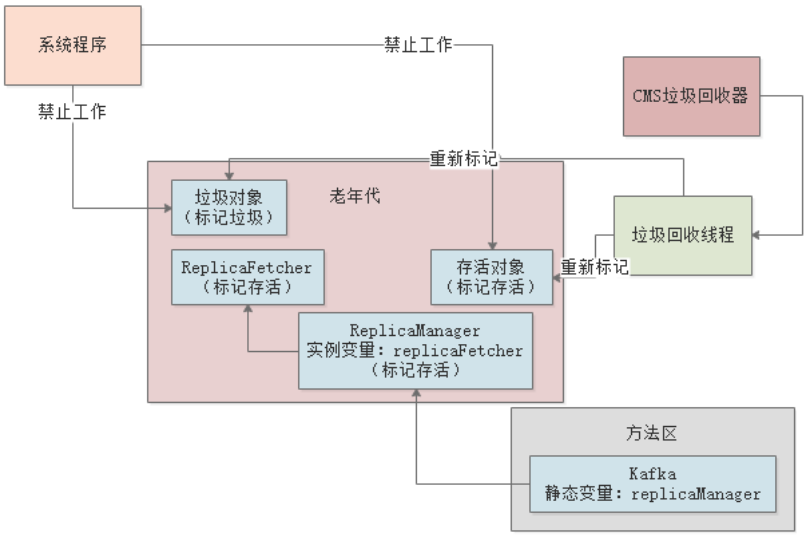
因为第二阶段里，你一边标记存活对象和垃圾对象，一边系统在不停运行创建新对象，让老对象变成垃圾

所以第二阶段结束之后，绝对会有很多存活对象和垃圾对象，是之前第二阶段没标记出来的，如下图。



所以此时进入第三阶段，要继续让系统程序停下来，再次进入“Stop the World”阶段。

然后重新标记下在第二阶段里新创建的一些对象，还有一些已有对象可能失去引用变成垃圾的情况，如下图。

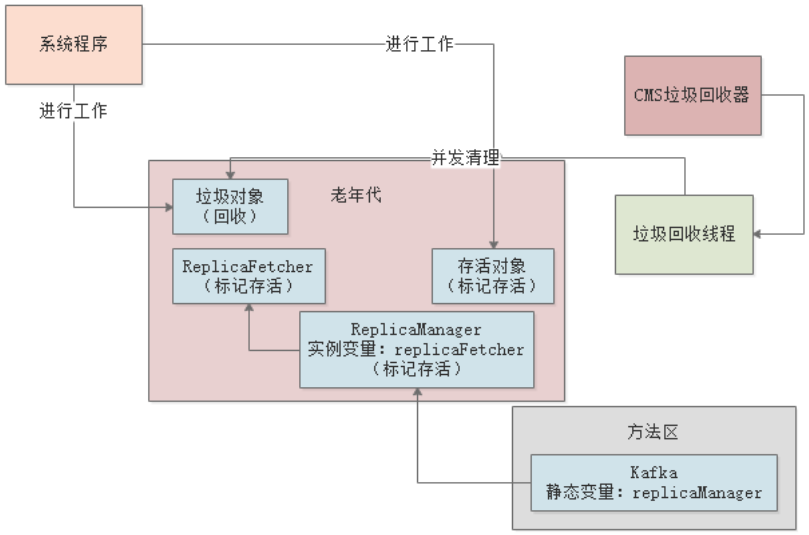


这个重新标记的阶段，是速度很快的，他其实就是对在第二阶段中被系统程序运行变动过的少数对象进行标记，所以运行速度很快。

接着重新恢复系统程序的运行，进入第四阶段：**并发清理**

这个阶段就是让系统程序随意运行，然后他来清理掉之前标记为垃圾的对象即可。

这个阶段其实是很耗时的，因为需要进行对象的清理，但是他也是跟系统程序并发运行的，所以其实也不影响系统程序的执行，如下图。



5、对CMS的垃圾回收机制进行性能分析

其实大家看完CMS的垃圾回收机制之后，就会发现，他已经尽可能的进行了性能优化了

因为最耗时的，其实就是对老年代全部对相关进行GC Roots追踪，标记出来到底哪些可以回收，然后就是对各种垃圾对象从内存里清理掉，这是最耗时的。

但是他的第二阶段和第四阶段，都是和系统程序并发执行的，所以基本这两个最耗时的阶段对性能影响不大。

只有 第一个阶段和第三个阶段是需要“Stop the World”的，但是这两个阶段都是简单的标记而已，速度非常的快，所以基本上对系统运行响应也不大。

明天的文章，我们就继续深入来看看CMS垃圾回收机制的各种细节以及一些参数一般如何设置。

6、昨日思考题

昨天的思考题，是一个学员真实面试中遇到的一个面试题：parnew+cms的gc，如何保证只做ygc，jvm参数如何配置?

该学员的回答:

加大分代年龄，比如默认15加到30;
修改新生代老年代比例，比如新生代老年代比例改成2:1
修改e区和s区比例，比如改成6:2:2

其实让大家去梳理这个思路，就是希望大家多一些思考，多一些梳理和总结。答案就在我们之前讲过的案例里，大家只要结合那个案例分析，就知道解答这面试题的思路。

大家可以尝试着作答，将自己的答案发至评论区。

7、今日思考题

看完了新生代和老年代的垃圾回收机制之后，大家来思考一下：为什么老年代的垃圾回收速度会比新生代的垃圾回收速度慢很多倍？到底慢在哪里？

明天的文章我们会深入探讨CMS垃圾回收机制和对应的参数，同时解答这个问题。

End

如何加群?

1、添加微信号：Giotto1245 （微信名：Jarvis）

2、发送 Jvm专栏的购买截图

3、人工操作，发送截图后请耐心等待被拉群

最后提醒：之前加过面试群的同学就不要重复加了

常见问题解答：

一、 如何生成自己的分享海报并获取返现?

方式1：

点击文章右上角**邀请好友**（如下图），生成自己的专属海报。

将海报发送给好友或分享朋友圈，朋友通过扫描你分享的海报购买课程，你将**获取返现24元**，可在个人中心中提现：

累计邀请30人，你将升级为高级推广员，此后每成功邀请一位朋友，返现翻倍。换句话说，从第31人开始，每成功邀请一位朋友，你将**获取返现48元**